Karlsruher Institut für Technologie

Programmieren – Wintersemester 2014/15
Software-Design und Qualität (SDQ)
`https://sdqweb.ipd.kit.edu/wiki/Programmieren`
Prof. Dr. Ralf H. Reussner · Kiana Rostami · Philipp Merkle

Abschlussaufgabe 2

Ausgabe: 02.03.2015
Abgabe: 30.03.2015 – 13:00

Allgemeine Hinweise

- Achten Sie darauf nicht zu lange Zeilen, Methoden und Dateien zu erstellen[1]
- Programmcode muss in englischer Sprache verfasst sein
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen der Pakete `java.lang`, `java.io` und `java.util`.
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1]
- Halten Sie alle Whitespace-Regeln ein[1]
- Halten Sie die Regeln zu Variablen-, Methoden und Paketbenennung ein und wählen Sie aussagekräftige Namen[1]
- Halten Sie die Regeln zu Javadoc-Dokumentation ein[1]
- Nutzen Sie nicht das default-Package[1]
- Achten Sie auf die korrekte Sichtbarkeit Ihrer Klassen[1]
- Halten Sie auch alle anderen Checkstyle-Regeln ein

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Montag, den 16. März**, freigeschaltet. Die Lösung dieses Blattes wird aus mehreren Java-Dateien bestehen. Laden Sie diese im Abgabezeitraum vom 16. März - 30. März 13 Uhr im Praktomaten hoch.

Beachten Sie, dass wir für die Abschlussaufgaben einen separaten Praktomen einrichten, den Sie über `https://praktomat.cs.kit.edu/` erreichen können. **Es ist nicht möglich, die Abschlussaufgaben im bisherigen Praktomaten hochzuladen.**

Es wird empfohlen, eine .zip-Datei (alternativ .tar oder .tar.gz) hochzuladen, welche die Ordnerstruktur widerspiegelt.

Checkstyle

Denken Sie daran, den Checkstyle-Regelsatz von unserer Homepage zu beziehen. Planen Sie, wie immer, ausreichend Zeit für die Abgabe ein, sollte der Praktomat Ihre Abgabe wegen einer Regelverletzung ablehnen.

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.

Terminal-Hilfsklasse

Laden Sie die `Terminal`-Klasse[2] von unserer Homepage herunter und platzieren Sie diese im Package `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die `Terminal`-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`.

Fehlerbehandlung

Ihr Programm muss auf alle ungültigen Benutzereingaben mit einer aussagekräftigen Fehlermeldung reagieren. Fehlermeldungen werden per `Terminal.printLine()` ausgegeben und müssen aus technischen Gründen unbedingt mit „`Error,`“ beginnen. Eine Fehlermeldung führt nicht dazu, dass das Programm beendet wird; es sei denn, die nachfolgende Aufgabenstellung verlangt dies ausdrücklich. Achten Sie insbesondere auch darauf, dass unbehandelte `RuntimeExceptions`, bzw. Subklassen davon—sogenannte *Unchecked Exceptions*—nicht zum Abbruch des Programms führen.

Langton-Ameisen (20 Punkte)

In dieser Aufgabe programmieren Sie eine Abwandlung der Langton-Ameise[3], die 1986 von Christopher Langton beschrieben wurde. Eine Langton-Ameise bewegt sich mittels einfacher Zugregeln über ein Spielfeld bestehend aus quadratischen Zellen. In der Ausgangssituation besteht das Spielfeld aus weißen quadratischen Zellen und die Ameise befindet sich auf einem der weißen Zellen. Außerdem hat die Ameise immer eine bestimmte Ausrichtung relativ zum Spielfeld. Anschließend bewegt sich die Ameise in dieser Richtung eine Zelle nach vorne. Betritt die Ameise eine weiße Zelle, so färbt sie die Zelle schwarz und die Ameise dreht sich 90 Grad im Uhrzeigersinn. Ist die betretene Zelle hingegen schwarz, so färbt sie die Zelle weiß und dreht sich 90 Grad entgegen dem Uhrzeigersinn. Durch die Umfärbung der Zellen durch die Ameise ergibt sich auf dem Spielfeld ein Muster, das abhängig von der Anzahl der Spielzüge sowohl chaotisch als auch regelmäßig erscheinen kann. Die Ameise bildet nach einer großen Anzahl an Schritten ein wiederkehrendes Muster, das einer Straße ähnelt und sich aus einer Menge von sich immer wiederholenden Zügen zusammensetzt. Folgende Abbildung illustriert die ersten zehn Züge einer Langton-Ameise. Die Ameise und ihre Blickrichtung ist durch einen Pfeil dargestellt.

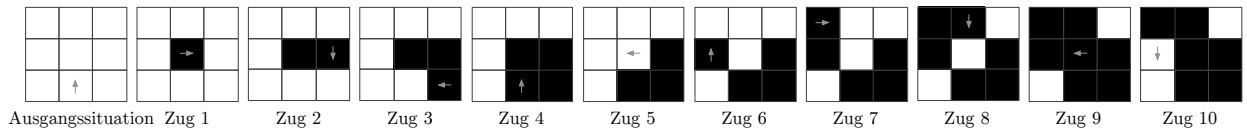

Abbildung 1: Die ersten zehn Züge einer Langton-Ameise

Im Rahmen dieser Aufgabe generalisieren wir die Langton-Ameise wie folgt:

- eine Zelle kann mehr als zwei Farben annehmen
- als Hindernis ausgezeichnete Zellen dürfen nicht betreten werden
- mehrere Ameisen können sich auf dem Spielfeld befinden
- Ameisen können sich in ihrem Zugverhalten unterscheiden

[2] `https://sdqweb.ipd.kit.edu/lehre/WS1415-Programmieren/Terminal.java`
[3] Siehe hierzu `http://de.wikipedia.org/wiki/Ameise_(Turingmaschine)`

A Farbiges Spielfeld mit Hindernissen

Das Spielfeld besteht aus quadratischen Zellen, die unterschiedliche Farben haben können oder einem Hindernis entsprechen können. Die Ränder des Spielfeldes sind offen; außerhalb des Spielfeldes ist die Welt undefiniert. Dies bedeutet, dass eine Ameise, die das Spielfeld verlässt, gleichzeitig auch das Spiel verlässt.

A.1 Zelle

Eine beliebige Zelle c_{ij} hat 8 Nachbarn, wie man Abbildung 2 entnehmen kann. Dabei bezeichnet i die Zeilennummer und j die Spaltennummer einer Zelle.

c_{i-1j-1}	c_{i-1j}	c_{i-1j+1}
c_{ij-1}	c_{ij}	c_{ij+1}
c_{i+1j-1}	c_{i+1j}	c_{i+1j+1}

Abbildung 2: Eine Zelle samt ihrer benachbarten Zellen

Auf einer Zelle, die kein Hindernis darstellt, kann sich maximal eine Ameise befinden. Zellen, die Hindernisse darstellen, werden niemals durch eine Ameise betreten.

A.1.1 Farben

Jede Zelle hat eine bestimmte Farbe. In dieser Aufgabe gibt es genau 5 Farben. Diese Farben sind beginnend mit 0 durchnummeriert. Die erste Farbe entspricht also der Farbe mit der Nummer 0 und die letzte (bzw. fünfte) Farbe entspricht der Farbe mit der Nummer 4. Nachdem eine Ameise eine Zelle betreten hat, lässt sich die neue Farbe der Zelle mit der Formel $f(x) = (4x + 23)\ \mathbf{mod}\ 5$ neu berechnen, wobei x und $f(x)$ jeweils die entsprechende Zahl für die aktuelle Farbe und die neue Farbe bezeichnen. **mod** stellt die Modulo-Operation dar. Beispielsweise wird mit Hilfe dieser Funktion die Farbe 0 in Farbe 3 umgewandelt.

A.1.2 Hindernisse

Nicht jede Zelle kann durch eine Ameise betreten werden. So eine Zelle wird auch als Hindernis bezeichnet. Stellt die nächste Zelle, die eine Ameise betreten möchte, ein Hindernis dar, so bleibt die Ameise während des aktuellen Schrittes in der aktuellen Zelle stehen und dreht sich in der aktuellen Zelle basierend auf ihrer Farbe und der Regel. Dies bedeutet, dass die aktuelle Zelle die eigentliche Hinderniszelle emuliert.

A.2 Ausrichtung der Ameise

Die Ameise hat immer eine Ausrichtung, die einer der Himmelsrichtungen entspricht und ist relativ zum Spielfeld angegeben. In dieser Aufgabe kann eine Ameise 8 verschiedene Ausrichtungen haben: Norden = `N`, Osten = `O`, Süden = `S`, Westen = `W`, Nordost = `NO`, Südost = `SO`, Südwest = `SW` und Nordwest = `NW`.

A.3 Bewegungsrichtung der Ameise

Die Ameise kann sich von der aktuellen Richtung aus in 4 verschiedenen Richtungen drehen: 45° nach rechts = 45, 90° nach rechts = 90, 45° nach links = 315, 90° nach links = 270. Die Bewegungsrichtung ist relativ zur aktuellen Ausrichtung der Ameise angegeben. In welcher dieser Richtungen sich die Ameise dreht, wird durch die Farbe der Zelle, die durch die Ameise betreten wird, bestimmt. Diese Regeln werden in Abschnitt B näher erläutert.

B Regel

Die Regeln geben die Beziehung zwischen den Zellfarben und den Drehrichtungen an. Eine Regel ist eine Sequenz der Elemente der Menge {45, 90, 270, 315}, die jeweils durch einen Bindestrich separiert sind. Die Länge einer Regel entspricht der Anzahl der Zellfarben. Somit gibt die Regel "45-270-90-90-315" in einem Spiel mit 5 Farben an, dass die Ameise bei der ersten Farbe eine Umdrehung um 45 Grad nach rechts, bei der zweiten Farbe eine Umdrehung um 90 Grad nach links, bei der dritten und vierten Farbe eine Umdrehung um 90 Grad nach rechts und bei der letzten Farbe eine Umdrehung um 45 Grad nach links macht.

C Spielzug

Jeder Spielzug besteht aus mehreren Schritten. In jedem Zug gibt es pro Ameise einen Schritt. Jeder Schritt umfasst eine Vorwärtsbewegung um eine Zelle (sofern kein Hindernis betreten wird) durch die jeweilige Ameise und ihre anschließende Drehung auf der neuen Zelle. Jeder Schritt kann von Ameise zu Ameise abweichend definiert sein. Somit ist ein Zug nur dann vollständig, wenn alle anwesenden Ameisen auf dem Spielfeld jeweils ihre Schritte vervollständigt haben. Siehe hierzu Abschnitt D.

D Ameise

Auf dem Spielfeld muss mindestens eine Ameise existieren. Insgesamt gibt es drei verschiedene Ameisentypen:

D.1 Standardameise

Die Standardameise hält sich genau an obige Regeln und tätigt pro Spielzug jeweils nur eine Vorwärtsbewegung (sofern kein Hindernis betreten wird) und eine anschließende Drehung entsprechend der aktuellen Regel, d.h. entweder 45°, 90°, 270°, oder 315° im Uhrzeigersinn. Bei insgesamt 5 Zellfarben und der Regel "90-45-315-90-270" dreht sich die Ameise bei der ersten Farbe in einem Schritt einmal nach rechts, d.h. 90° im Uhrzeigersinn, bei der zweiten Farbe in einem Schritt nur 45° im Uhrzeigersinn, bei der dritten Farbe in einem Schritt um 45° im Gegenuhrzeigersinn, bei der vierten Farbe in einem Schritt um 90° im Uhrzeigersinn und bei der letzten Farbe in einem Schritt einmal nach links, d.h. 90° im Gegenuhrzeigersinn.

D.2 Sportliche Ameise

Eine sportliche Ameise zeichnet sich durch ihre besonders hohe Geschwindigkeit aus, indem sie n mal schneller als eine Standardameise ist. Jeder Schritt einer sportlichen Ameise besteht aus n Zwischenschritten. Jeder Zwischenschritt wiederum besteht wie bei einem Schritt der Standardameise aus einer Vorwärtsbewegung um eine Zelle (sofern kein Hindernis betreten wird) und der anschließenden Drehung. Die Zahl n wird durch die Programmeingabe bestimmt, siehe hierzu Abschnitt F.4.

D.3 Faule Ameise

Eine faule Ameise hingegen spielt nur einen Schritt pro n Züge, indem sie zunächst im ersten Zug einen Schritt spielt und während der nächsten $n-1$ Züge sich nicht mehr auf dem Spielfeld bewegt. Sie wiederholt anschließend dieses Muster bis zum Ende des Spiels. Bei der Zahl n handelt es sich um dieselbe Zahl n wie bei der sportlichen Ameise, siehe hierzu auch Abschnitt F.4.

E Kollision

Was passiert, wenn mehrere Ameisen innerhalb eines Spielzuges dieselbe Zelle betreten möchten? Welche Ameise hat den Vorrang? Um solche Kollisionen zu verhindern, ordnen wir jeder Ameise einen eindeutigen kleinen Buchstaben von a bis z zu. Es kann somit maximal 26 gleichzeitig anwesende Ameisen auf dem Spielfeld geben. Die Ameisen ziehen dann in alphabetisch aufsteigender Reihenfolge, wodurch Ameisen mit kleinerem Buchstaben

Vorrang haben. Ist eine Zelle, die eine Ameise betreten möchte, bereits durch eine andere Ameise besetzt, wird die besetzte Zelle wie ein Hindernis behandelt, auch wenn die Zelle zu Beginn des aktuellen Spielzuges noch leer war.

F Kommandozeilenparameter

Beim Start erwartet Ihr Programm als **erstes Kommandozeilenargument** einen Pfad auf eine Textdatei. Diese Textdatei beinhaltet das Spielfeld inklusive der aktuellen Ausrichtung der Ameise(n). Zum Einlesen der Datei dürfen Sie die bereitgestellte Klasse `FileInputHelper`[4] nutzen. Im Gegensatz zur `Terminal`-Klasse darf—und muss—die `FileInputHelper`-Klasse in den Praktomaten hochgeladen werden, falls sie eingesetzt wird.

Als **zweites und drittes Kommandozeilenargument** kann die *Regel* und der *Beschleunigungsfaktor* bestimmt werden. Diese beiden Kommandozeilenargumente sind optional.

F.1 Aufbau des Spielfeldes

Die Spieldatei besteht aus einer oder mehreren Zeilen. Jede Zeile entspricht einer Zeile des Spielfeldes und besteht somit aus einer Folge von Zellen. Eine Zelle kann entweder leer sein, durch eine Ameise besetzt sein, oder ein Hindernis repräsentieren.

Eine *leere Zelle* wird durch ihre Farbe dargestellt, also eine Zahl aus der Menge {0, 1, 2, 3, 4, 5}.

Eine *Zelle mit einer Ameise* wird durch einen Buchstaben dargestellt. Ein Kleinbuchstabe bedeutet, dass die Ameise in der Ausgangsituation nach unten – Süden = S – schaut. Ein Großbuchstabe bedeutet, dass die Ameise in der Ausgangssituation nach oben – Norden = N – schaut. Die Farbe der initialen Position einer Ameise in der Ausgangssituation entspricht stets 0. Dies entspricht dem Schritt 0 in der Abbildung 1.

Ein *Hindernis* wird durch das Zeichen `*` dargestellt.

Alle Zeilen der Spieldatei haben dieselbe Länge. Die Datei darf außerdem keine Leerzeichen enthalten.

Tritt beim Einlesen der Datei ein Fehler auf, sei es ein Fehler beim Lesen der Datei, ein Syntaxfehler, oder ein semantischer Fehler, so wird eine Fehlermeldung ausgegeben und das Programm beendet sich mittels `System.exit(1)`. Ein Syntaxfehler besteht, wenn die Datei nicht gemäß obenstehender Spezifikation geformt ist. Ein semantischer Fehler besteht, wenn sich z.B. mehrere Ameisen mit demselben Buchstaben auf dem Spielfeld befinden.

F.2 Beispiel einer Spieldatei

Folgendes Beispiel präsentiert eine wohlgeformte Spieldatei zum Aufbau des in Abbildung 1 dargestellten Spielfeld in der Ausgangssituation:

```
000
000
0F0
```

Das in Abbildung 1 abgebildete Spielfeld in der Ausgangssituation kann mit unterschiedlichen Eingaben erzeugt werden, z.B. durch Auswahl eines anderen Großbuchstabens für die Ameise oder durch Auswahl unterschiedlicher Zellfarben. Die obige Eingabe ist somit nur eine Beispieleingabe für den Aufbau dieses Spielfeldes.

[4] `https://sdqweb.ipd.kit.edu/lehre/WS1415-Programmieren/FileInputHelper.java`

F.3 Regel

Existiert ein Kommandozeilenargument `rule=`*value*, so bestimmt *value*, wie die Ameise sich auf einer bestimmten Farbe drehen muss. *value* ist ein Platzhalter für eine Regel, die aus genau 5 Bewegungsrichtungen, jeweils separiert durch einen Bindestrich (-), bestehen muss. Die Bewegungsrichtungen in einer Regel stammen aus der Menge {45, 90, 270, 315}, siehe hierzu Abschnitt B. Bei fehlender Angabe des Parameters `rule=`*value* ist der Standardwert hier `270-90-315-45-90`.

Tritt beim Einlesen der Regel ein Syntaxfehler auf, so wird eine Fehlermeldung ausgegeben und das Programm beendet sich mittels `System.exit(1)`. Ein Beispiel für einen Syntaxfehler bildet die Regel, die mehr oder weniger als 5 Zeichen hat oder andere Buchstaben außer {45, 90, 270, 315} enthält.

F.4 Beschleunigungsfaktor

Existiert ein Kommandozeilenargument `speedup=`*value*, so bestimmt *value*, um welchen Faktor eine sportliche Ameise schneller, bzw. eine faule Ameise langsamer als eine Standardameise ist. *value* ist ein Platzhalter für eine positive Integerzahl ab 1. `speedup=1` entspricht dem Fall, wenn alle auf dem Spielfeld anwesenden sportlichen und faulen Ameisen sich genauso schnell wie eine Standardameise fortbewegen. `speedup=n` gibt bei einer sportlichen Ameise an, dass sie pro Schritt einer Standardameise n Unterschritte machen muss. Im Gegensatz gibt `speedup=n` bei einer faulen Ameise an, wie viele Spielzüge sie überspringen muss, während die Standardameise einen Schritt pro Spielzug macht (wobei mit `n=1` keine Spielzüge übersprungen werden). Bei fehlender Angabe des Parameters `speedup=`*value* ist der Standardwert hier `speedup=2`.

Tritt beim Einlesen der Regel ein Syntaxfehler auf, so wird eine Fehlermeldung ausgegeben und das Programm beendet sich mittels `System.exit(1)`. Ein Beispiel für einen Syntaxfehler bildet eine negative Zahl für den Beschleunigungsfaktor.

G Interaktive Benutzerschnittstelle

Nach dem Start nimmt Ihr Programm über die Konsole mittels `Terminal.readLine()` neun Arten von Befehlen entgegen. Nach Abarbeitung eines Befehls wartet das Programm auf weitere Befehle, bis das Programm irgendwann durch `quit` beendet wird. Alle Befehle werden auf dem aktuellen Zustand des Spiels ausgeführt.

Tritt bei folgenden Befehlen ein Fehler auf, so wird eine Fehlermeldung ausgegeben. Das Programm beendet sich jedoch nicht und wartet auf weitere Befehle.

G.1 Der `move`-Befehl

Der Befehl `move value` führt die angegebene Anzahl an Spielzügen durch. Der Platzhalter *value* bestimmt, wie viele Spielzüge ab der aktuellen Position der jeweiligen Ameise gemacht werden müssen. *value* ist eine positive Integerzahl inklusive 0. `move 0` entspricht dem aktuellen Spielzustand. Dies bedeutet, dass keine Spielzüge gespielt werden. Achten Sie darauf, dass der aktuelle Zustand nicht zur Anzahl der Spielzüge zählt. Somit spielt die Ameise in der Abbildung 1 nur 10 Spielzüge.

Nach diesem Befehl folgt keine Konsolenausgabe.

G.2 Der `print`-Befehl

Der `print`-Befehl gibt das aktuelle Spielfeld aus.

G.2.1 Ausgabeformat

Das Spielfeld wird zeilenweise ausgegeben. Eine leere Zellen wird durch ihre Farbe aus der Menge {`0`,`1`,`2`,`3`,`4`} ausgegeben. Eine Zelle mit einer Ameise wird durch einen Buchstaben in Kleinschreibung dargestellt. Die Richtung

der Ameisen wird nicht ausgegeben. Eine Zelle mit einem Hindernis wird durch das Zeichen * dargestellt.

G.2.2 Beispiel

Im Folgenden wird ein Beispiel eines wohlgeformten Spielfeldes dargestellt:

```
1*0
3e4
12*
```

G.3 Der `position`-Befehl

Der Befehl `position ant` gibt die Position der Ameise *ant* auf dem Spielfeld aus. *ant* ist dabei ein Platzhalter für eine bestimmte Ameise, d.h. einen Buchstaben zwischen a und z. Groß- und Kleinschreibung für *ant* wird bei diesem Befehl ignoriert.

Das Spielfeld wird in diesem Fall wie eine Matrix behandelt. Dies bedeutet, dass die Zeilen von oben nach unten und die Spalten von links nach rechts beginnend mit 0 durchnummeriert sind.

G.3.1 Ausgabeformat

Die Ausgabe erfolgt in nur einer Zeile. Zunächst wird die Zeilennummer und anschließend die Spaltennummer ausgegeben. Die Zeilen- und Spaltennummer sind durch ein Komma separiert. Es werden keine weiteren Zeichen ausgegeben.

G.3.2 Beispiel

Nach dem zehnten Zug des in Abbildung 1 dargestellten Spieles wird die aktuelle Position der Ameise wie folgt ausgegeben:

```
1,0
```

G.4 Der `field`-Befehl

Der Befehl `field x,y` gibt den Zustand der Zelle mit Koordinaten x (Zeilennummer des Feldes beginnend mit 0) und y (Spaltennummer des Feldes beginnend mit 0) aus. Die Zeilen- und Spaltennummer sind durch ein Komma separiert. Wie bereits beschrieben sollen Sie bei der Zeilen- und Spaltennummer sich an einer Matrix orientieren.

G.4.1 Ausgabeformat

Existiert die Zelle, deren Koordinaten vorgegeben ist, so kann sie sich in einem der folgenden Zustände befinden:

- Eine leere Zelle wird durch ihre Farbe dargestellt. In diesem Fall wird eine Zahl $\in\{0,1,2,3,4\}$ ausgegeben.
- Eine Zelle mit einer Ameise wird durch den zugehörigen Buchstaben in Kleinschreibung dargestellt.
- Eine Zelle mit einem Hindernis wird durch einen Stern * dargestellt.

Es findet keine weitere Ausgabe statt.

G.4.2 Beispiel

Im Folgenden finden Sie zu obigen Fällen jeweils ein Beispiel:

Bei Anwesenheit der Ameise e:

```
e
```

Bei einem Hindernis:

```
*
```

Bei der Farbe mit Markierung 3:

```
3
```

G.5 Der `direction`-Befehl

Der Befehl `direction ant` gibt die Ausrichtung der Ameise *ant* auf dem Spielfeld aus. *ant* ist dabei ein Platzhalter für eine bestimmte Ameise. Groß- und Kleinschreibung für *ant* wird bei diesem Befehl ignoriert. Eine Ameise kann insgesamt 8 Himmelsrichtungen {Norden = N, Osten = O, Süden = S, Westen = W, Nordost = NO, Südost = SO, Südwest = SW und Nordwest = SW} haben.

G.5.1 Ausgabeformat

Die Ausgabe erfolgt durch die Angabe der Ameisenrichtung $\in$ {`N`, `O`, `S`, `W`, `NO`, `SO`, `SW`, `NW`}. Es werden keine weiteren Zeichen ausgegeben.

G.5.2 Beispiel

Nach dem zehnten Zug des in Abbildung 1 dargestellten Spieles wird die aktuelle Richtung der Ameise wie folgt ausgegeben:

```
S
```

G.6 Der `ant`-Befehl

Der `ant`-Befehl listet alle auf dem Spielfeld anwesenden Ameisen auf.

G.6.1 Ausgabeformat

Jede Ameise wird durch einen Buchstabe in Kleinschreibung dargestellt. Die Ausgabe der Ameisen erfolgt in alphabetisch aufsteigender Reihenfolge in einer Zeile. Zwei Ameisen sind stets nur durch ein Komma separiert. Es werden keine weiteren Zeichen ausgegeben.

G.6.2 Beispiel

Im Folgenden ist die Programmausgabe bei einem `ant`-Befehl exemplarisch für den Falls, dass sich 6 Ameisen gleichzeitig auf einem aktuellen Spielfeld befinden, dargestellt:

```
b,d,h,k,m,z
```

G.7 Der `create`-Befehl

Der Befehl `create ant,x,y` erzeugt die Ameise `ant` auf der Zelle mit den Koordinaten `x` (Zeilennummer des Feldes beginnend mit 0) und `y` (Spaltennummer des Feldes beginnend mit 0). Die Ameise kann bei der Erstellung nur eine der beiden folgenden Ausrichtungen haben: Ein Kleinbuchstabe bedeutet, dass die erzeugte Ameise nach unten – Süden = S – schaut. Ein Großbuchstabe bedeutet, dass die erzeugte Ameise nach oben – Norden = N – schaut. Achten Sie bei diesem Befehl besonders auf potentielle semantische/syntaktische Fehler, wie z.B.: Erzeugen einer neuen Ameise mit einem bereits existierenden Namen, Erzeugen einer Ameise außerhalb des Spielfeldes oder Erzeugen einer Ameise auf einer bereits besetzten Zelle.

Dieser Befehl verursacht keine Konsolenausgabe.

G.7.1 Beispiel

```
create e,1,1
```

G.8 Der `escape`-Befehl

Der Befehl `escape ant` sorgt dafür, dass die Ameise `ant` das Spielfeld sofort verlässt. *ant* ist dabei ein Platzhalter für eine Ameise. Groß- und Kleinschreibung für *ant* wird bei diesem Befehl ignoriert. Nachdem eine Ameise das Spielfeld verlassen hat, kann das Spiel, soweit es sich nicht um die letzte Ameise auf dem Spielfeld handelt, wie gewohnt fortgesetzt werden. Handelt es sich um die letzte Ameise auf dem Spielfeld, terminiert das Programm.

Dieser Befehl verursacht keine Konsolenausgabe.

G.8.1 Beispiel

```
escape e
```

G.9 Der `quit`-Befehl

Dieser Befehl beendet das Programm. Dabei findet keine Konsolenausgabe statt.