Kapitel 15 Eine Spielsteuerung als Spielleiter

Lernziele:

In diesem Kapitel werden **Zustandsdiagramme** vorgestellt. Sie helfen Abläufe zu planen. Weiterhin wird die **Mehrfachauswahl** vorgestellt, die Programme mit mehreren bedingten Anweisungen übersichtlicher und kürzer machen kann. Wiederholt wird Objektkommunikation und das Überladen des Konstruktors.

Hinweis:

Inhaltlich sind Kapitel 15 und Kapitel 16 eng miteinander verknüpft. So werden in Kapitel einige zu "erledigende Aufgaben" aufgezeigt, die dann erst in Kapitel 16 behandelt werden.

15.0 Grundwissen

In den bisherigen 14 Kapiteln wurden mehrere Diagrammtypen eingeführt und verwendet. Sinn und Zweck dieser Diagramme ist die Planung bzw. Veranschaulichung. Bevor nun in diesem Kapitel ein neuer Diagrammtyp hinzukommt, ist ein kleiner Rückblick sinnvoll.

Aufgabe 15.0
Schreibe alle Diagrammtypen auf, die bisher verwendet wurden und gib jeweils den Einsatzbereich an.

15.1 Was kann passieren, wenn sich Mampfi bewegt?

Aufgabe 15.1
Wenn Mampfi durch einen „Schritt“ eine neue Zelle erreicht hat, gibt es mehrere unterschiedliche Fälle.

a) Beschreibe möglichst knapp jeden der möglichen Fälle.
b) Im Spielablauf "passiert etwas", wenn Mampfi sich in eine neue Zelle bewegt. Dafür bist du als Programmierer des Spiels verantwortlich. Nenne die Aktionen, die nach einem Schritt von Mampfi ausgeführt werden müssen. Unterscheide dabei die Fälle aus Teilaufgabe a). Du darfst / sollst ruhig ein wenig vorausschauen, wie das Spiel am Ende ablaufen soll und musst dich nicht auf den aktuellen Stand des Projekts beschränken. Lasse jedoch an dieser Stelle die Monster außer betracht. Sie werden ab Kapitel 17 berücksichtigt.
   Notiere deine Antworten im Heft.

Wenn Mampfi sich bewegt und eine Zelle betritt, so kann diese leer sein, ein „normaler“ Krümel oder ein Powerkrümel darauf liegen. Im ersten Fall gibt es nichts zu tun. Liegt ein Krümel auf der Zelle muss entsprechend dem Punktewert des Krümels der Punktestand erhöht werden und der verspeiste Krümel muss verschwinden. Im Falle eines Powerkrümel muss sich auch die Verwundbarkeit von Mampfi und den Monstern ändern.
Graphisch darstellen lassen sich die eben beschriebenen Abläufe in einem **Zustandsdiagramm**:

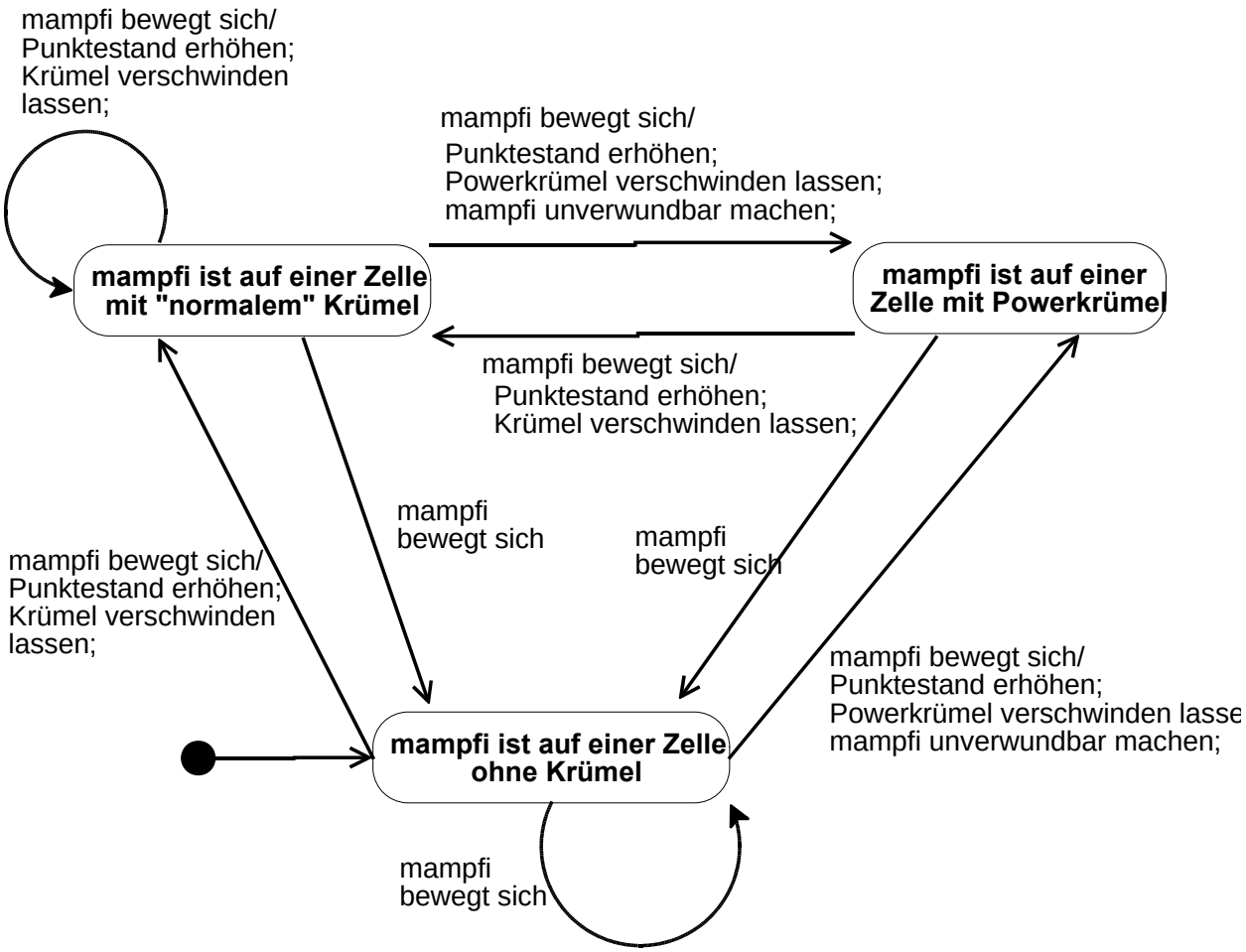


Abbildung 1: Zustandsdiagramm zur Planung des Spielablaufs - es wird deutlich welche unterschiedlichen Spielsituationen es gibt und welche Aktionen die Spielsteuerung bei der Bewegung von Mampfi abhängig von der Spielsituation durchführen muss

Jeder **Zustand** wird durch ein abgerundetes Rechteck dargestellt und aussagekräftig beschriftet. Übergänge von einem Zustand zu einem anderen werden durch einen Pfeil symbolisiert (**Zustandsübergang**). Der Pfeil ist mit dem Ereignis beschriftet, das den Übergang auslöst. In unserem Fall sind die Ereignisse Methodenaufrufe, die Mampfi in Bewegung bringen, d.h. *NachNordenGehen*, *NachWestenGehen* usw.. Abkürzend ist in dem Zustandsdiagramm oben nur „Mampfi bewegt sich“ notiert.
Optional können auf dem Zustandsübergang - getrennt durch einen Schrägstrich - Aktionen geschrieben werden, die durch den Übergang ausgelöst werden. In dem Beispiel oben gehört „den

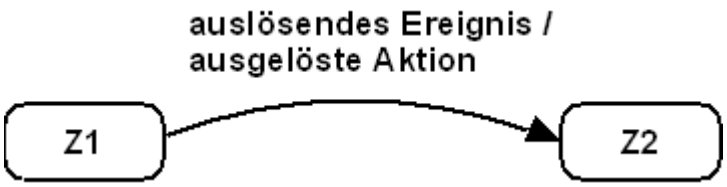


Abbildung 2: Zustandsdiagramm allgemein – Zwei Zustände und ein beschrifteter Zustandsübergang.

Punktestand erhöhen“ dazu. In der Abbildung 2 sind die Bestandteile eines Zustandsdiagramms nochmals dargestellt.

Aufgabe 15.2
Wer überprüft, ob auf der Zelle von Mampfi ein Krümel ist? Wer kümmert sich darum, dass gegebenenfalls der Punktestand erhöht wird? Wo wird der Punktestand angezeigt?

15.2 Die Klasse SPIELSTEUERUNG

Es gibt mehrere Möglichkeiten den Spielverlauf zu steuern, d.h. Aufgaben wie die Erhöhung des Punktestands zu übernehmen. Hier wird eine Lösungsvariante vorgestellt, in der die Kontrolle über eine Klasse SPIELSTEUERUNG erfolgt.
Eine zentrale Spielsteuerung in der Hand eines Objekts kann dann auch weitere "organisatorische Aufgaben" wie das Steuern des Spiels durch Tastatureingaben übernehmen (Kapitel 15.4).

Aufgabe 15.3
Überlege dir sinnvolle Attribute für die Klasse SPIELSTEUERUNG.

Die Spielsteuerung ist beispielsweise für die Aktualisierung des Punktestands verantwortlich. Dazu muss sie wissen, wo sich Mampfi befindet und ob auf der entsprechenden Position ein Krümel liegt. Die Spielsteuerung muss folglich mit dem Labyrinth und Mampfi kommunizieren. Dies ist nur möglich, wenn Beziehungen zwischen den Klassen existieren.

Aufgabe 15.4

a) Skizziere die beschriebenen Beziehungen in einem Klassendiagramm (ohne Attribute und Methoden). Beschrifte sinnvoll und achte darauf, dass die Richtung der Beziehung der Kommunikationsrichtung entspricht.
b) Welche Ergänzungen ergeben sich aus Teilaufgabe a) für die Klasse SPIELSTEUERUNG? Zeichne das erweiterte Klassendiagramm.

c) Bisher musste zum Testen jeweils das Labyrinth und Mampfi "per Hand durch Mausklicks" in der BlueJ-Umgebung erzeugt werden. Notiere die Methodenaufrufe, die zum Erzeugen dieser beiden Objekte nötig sind.
d) Die Klasse Spielsteuerung soll das Labyrinth und Mampfi automatisch erzeugen. An welcher Stelle der Klasse SPIELSTEUERUNG muss der Quelltext aus Teilaufgabe c) eingefügt werden?

Abbildung 3 zeigt Attribute und den Konstruktor der Klasse SPIELSTEUERUNG. Neben dem punkteStand sind weitere Attribute die Anzahl der Leben, die Mampfi noch hat, und das Level, in dem sich Mampfi aktuell befindet. Diese Attribute werden zwar in den nächsten Kapiteln noch nicht sofort benötigt, geben aber schon einen Ausblick, wie man das Spiel weiter gestalten kann.

Abbildung 3: Attribute und Konstruktor der Klasse SPIELSTEUERUNG

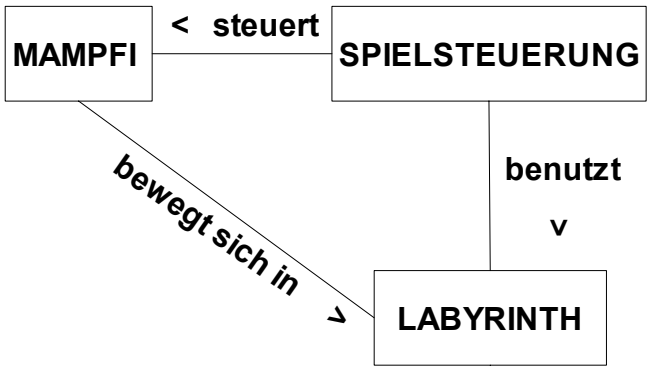


Abbildung 4: Beziehungen der Klasse MAMPFI, SPIELSTEUERUNG und LABYRINTH

Um die (neuen) Beziehungen der Klassen aus Abbildung 4 umzusetzen, benötigt die Klasse SPIELSTEUERUNG zwei Referenzattribute (Abbildung 5).

Abbildung 5: Ergänzung der Klasse SPIELSTEUERUNG um Referenzattribute

Aufgabe 15.5
Ergänze in deinem BlueJ Projekt die Klasse SPIELSTEUERUNG.
Achte beim Testen darauf, dass beim Erzeugen der Spielsteuerung automatisch das Labyrinth und Mampfi erzeugt werden (vgl. Aufgabe 15.4). Falls dir dies Schwierigkeiten bereitet findest du Tipps im Anhang.
Vergiss die Dokumentation in der neuen Klasse nicht! Überprüfe deine Dokumentation, indem du eine neue Version deiner Javadoc erstellst!

Abbildung 6 zeigt ein (mögliches) Anzeigefenster nach der Erzeugung eines Objekts der Klasse SPIELSTEUERUNG.

Aufgabe 15.6
Nenne zwei Punkte, die bei Anzeige in Abbildung 6 nicht gut sind und verbessert werden müssen.

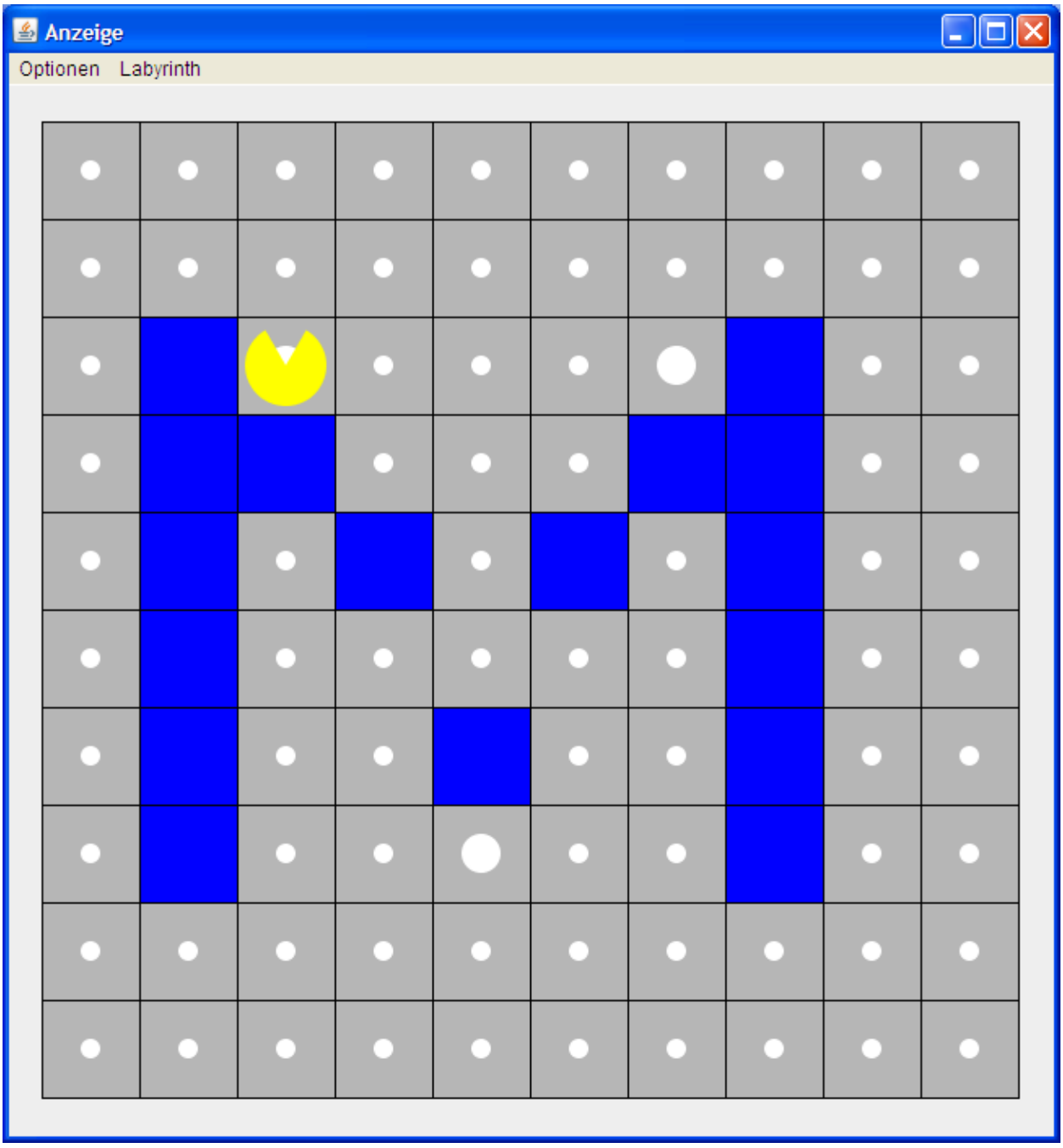


Abbildung 6: Anzeigefenster nach dem Erzeugen eines Objekts der Klasse SPIELSTEUERUNG

In Abbildung 6 sitzt Mampfi auf einem Feld mit einem Powerkrümel. Dies sollte nicht so sein! Es wäre sinnvoll, dass auf dem Startfeld von Mampfi kein Krümel liegt. Diese Aufgabe wird in Kapitel 16 gelöst werden.
Weiterhin ist es für ein Spiel unbefriedigend, dass weder der Punktestand noch die Anzahl der Leben und das Level angezeigt werden. Wie sich dies ändern lässt wird im jetzt folgenden Kapitel (15.3) erklärt.

15.3 Die Klasse STEUERUNGSANZEIGE

Passend zur Spielsteuerung gibt es eine Klasse STEUERUNGSANZEIGE, die es ermöglicht den aktuellen Punktestand, die Anzahl der Leben usw. anzuzeigen. Attribute und Methoden dieser Klasse sind in Abbildung 7 zusammengefasst.

Hinweis:
Die Klasse STEUERUNGSANZEIGE hat die gleiche Funktion wie die Klasse PUNKTELISTEANZEIGE bzw. wie die Symbolklassen.

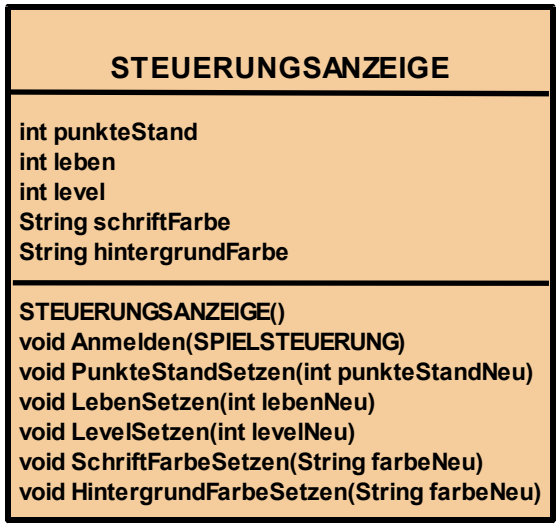


Abbildung 7:
Attribute und Methoden der Klasse STEUERUNGSANZEIGE

Aufgabe 15.7
Wie muss das Klassendiagramm aus Abbildung 4 bzw. die Klasse SPIELSTEUERUNG ergänzt werden, um die STEUERUNGSANZEIGE zu integrieren?

Es gibt eine Besonderheit bei der Steuerungsanzeige. Indirekt ermöglicht sie (in Zusammenspiel mit dem Backend) auch eine Tastatursteuerung, deren Details in der Spielsteuerung festgelegt werden (siehe Kapitel 15.4). Damit dies möglich ist, muss sich die Spielsteuerung bei der Steuerungsanzeige über die Methode *Anmelden* anmelden. Als Eingabewert muss die Spielsteuerung eine Referenz auf sich selbst übergeben. Dies ist mit Hilfe des Schlüsselworts `this` möglich. Somit muss im Konstruktor der Klasse SPIELSTEUERUNG nach dem Erzeugen des Objekts anzeige (von der Klasse STEUERUNGSANZEIGE) folgende Zeile stehen:

```
anzeige.Anmelden(this);
```

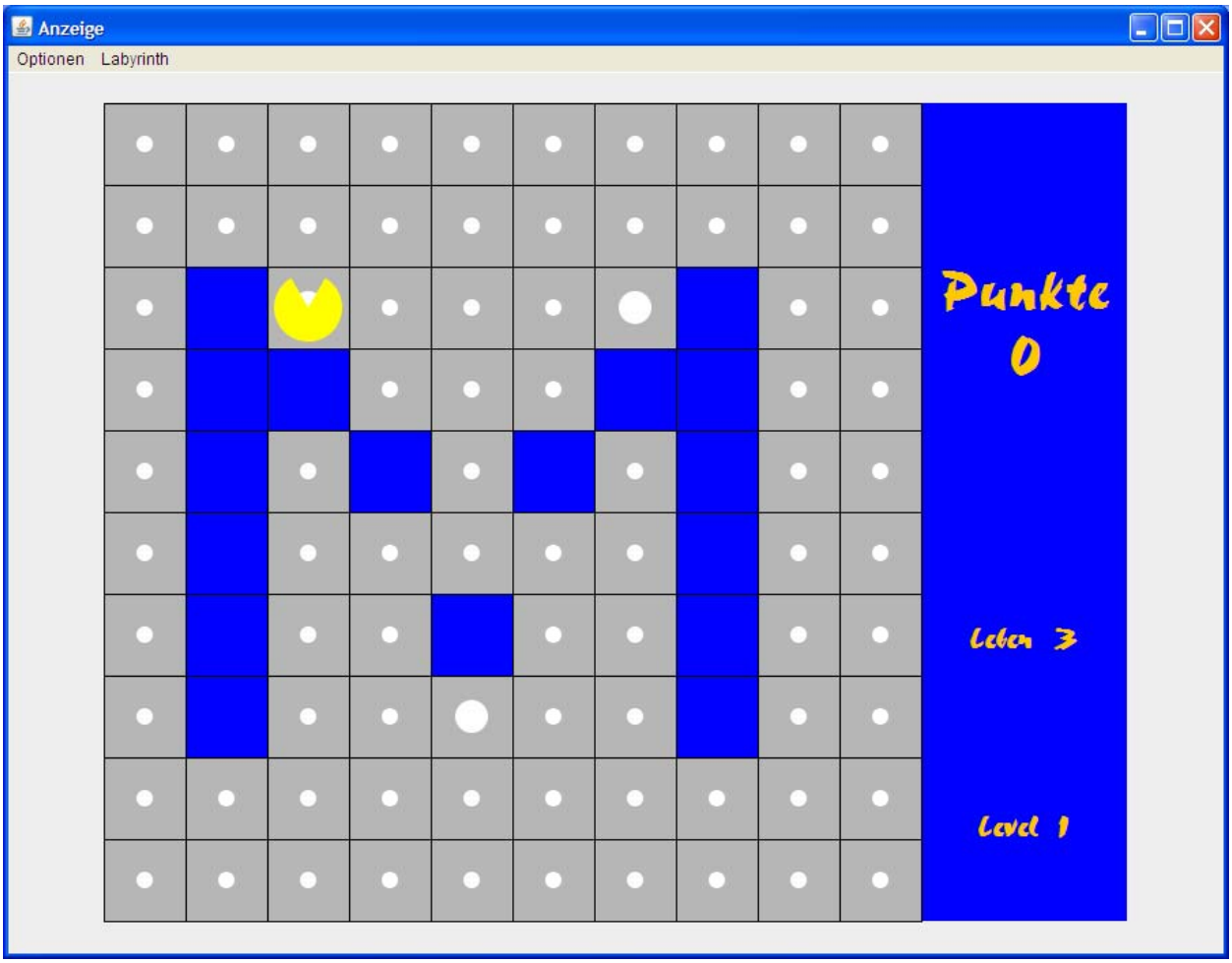
Aufgabe 15.8
a) Lade dir die Klasse STEUERUNGSANZEIGE herunter und integriere sie in dein BlueJ-Projekt (Bearbeiten --> Klasse aus Datei hinzufügen)
b) Setze in der Klasse SPIELSTEUERUNG die Beziehung zur Steuerungsanzeige um. Achte im Konstruktor auf sinnvolle Anfangswerte. Die Schnittstelle der STEUERUNGSANZEIGE zeigt Abbildung 7.
Ein mögliches Ergebnis beim Testen ist in Abbildung 8 dargestellt.

Hinweise:
- Wichtig ist im Konstruktor der Klasse SPIELSTEUERUNG nach dem Erzeugen der Steuerungsanzeige das Anmelden (s.o.).
- Es ist möglich, bei der Anzeige von Punkten, Leben und Level eine andere Schriftart[1] zu verwenden. Gibt es eine Datei mit dem Namen schriftart.ttf[2] im Projektordner, so wird diese geladen und verwendet. Falls keine solche Datei vorhanden ist, wird die Standardschrift deines Computers verwendet.
(Auf der Plattform steht eine Datei schriftart.ttf zur Verfügung, die du in deinen Pro-

1 engl.: font
2 die Extension ttf steht für True Type Font

jektordner speichern kannst. Möchtest du einen andere Schriftart verwenden, so kannst du Schriftarten im Internet oder bei Windows im Verzeichnis Windows\Fonts finden. Wird eine Schriftart-Datei vom Typ *.ttf in den Projektordner **kopiert**, muss man sie nur noch auf schriftart.ttf umbenennen.)

- Bei Schwierigkeiten können die Klassendiagramme in Abbildung 9 und 10 bzw. die Tipps im Anhang helfen.



r
Abbildung 8: Anzeigefenster, nun auch mit der Anzeige der Punkte, der Leben und des Levels

Die Beziehungen von Objekten der Klasse SPIELSTEUERUNG zu Objekten anderer Klassen ist in folgendem Klassendiagramm dargestellt:

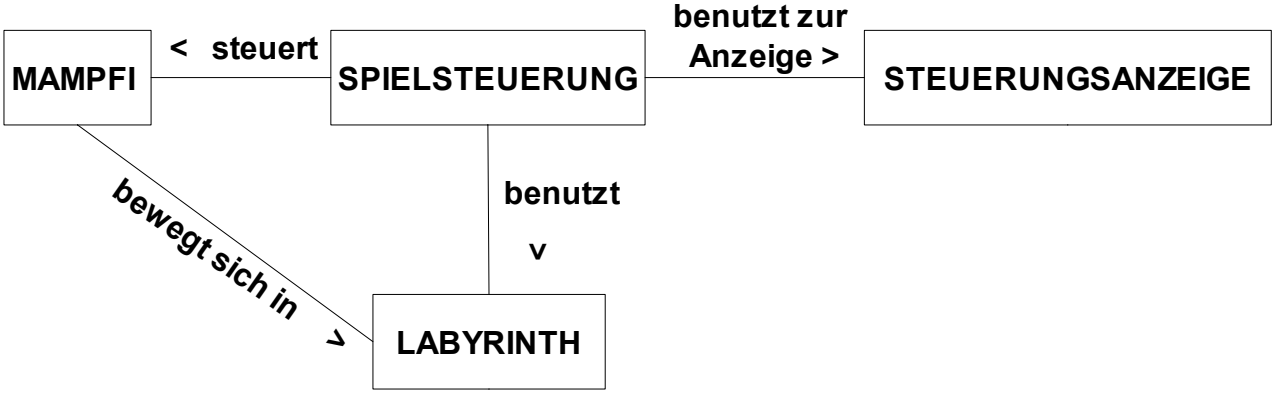


Abbildung 9: Beziehungen der Klasse SPIELSTEUERUNG

In der Klasse SPIELSTEUERUNG ist ein drittes Referenzattribut nötig, um die Beziehung zur Steuerungsanzeige umzusetzen (Abbildung 10)

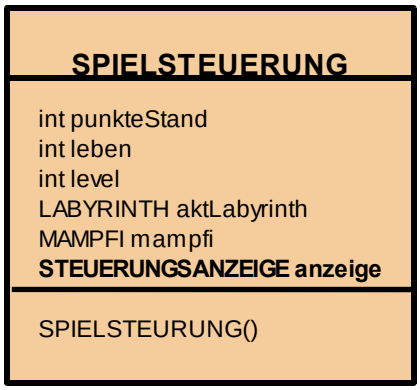


Abbildung 10: Die Klasse STEUERUNGSANZEIGE ergänzt um das Referenzattribut anzeige

15.4 Tastatursteuerung

Selbstverständlich ist es für ein Computerspiel nicht akzeptabel, die Methodenaufrufe für die Bewegungen von Mampfi über das Kontextmenü in der BlueJ Umgebung durchzuführen. Eine Steuerung über die Tastatur ist gefragt. Das Backend hilft dabei: Gibt es in der Klasse SPIELSTEUERUNG eine Methode ***AufTasteReagieren***, so sorgt das Backend dafür, dass bei jedem Ereignis „Tastendruck“ diese Methode *AufTasteReagieren* ausführt wird.

Hinweis:

- Das Backend kennt die Spielsteuerung, weil sie über die Steuerungsanzeige angemeldet wurde (siehe Methodenaufruf *anzeige.Anmelden(this)* in Kapitel 15.3).
- Die Tastatursteuerung erfolgt vollautomatisch durch das Backend. D.h. es ist **nicht** nötig, in einer unserer Klassen die Methode *AufTasteReagieren* der Steuerungsanzeige aufzurufen, sondern das Backend macht dies[3]. Da der Aufruf der Methode *AufTasteReagieren* im Backend schon implementiert ist, bevor wir die Klasse SPIELSTEUERUNG in Java programmieren, muss die Methode wie im Backend vorgesehen ***AufTasteReagieren*** heißen. Alternative Namen und Schreibweisen sind **nicht** möglich.

Aufgabe 15.9
Welche wichtige Information benötigt die Methode *AufTasteReagieren* als Eingabewert? Wie würdest du diese Information speichern? Als Zahl, als Zeichen, ...?

3 Für Experten:
Die Kommunikation bei der Tastatursteuerung erfolgt nicht vom Frontend (unsere Klassen) zum Backend, sondern in die umgekehrte Richtung. Das Backend "überwacht" die Tastatur. Wird eine Taste gedrückt, ruft das Backend die Methode *AufTasteReagieren* bei Objekten der Klasse SPIELSTEUERUNG auf.
Selbstverständlich benötigt das Backend als Voraussetzung für die Kommunikation eine Referenz auf die Spielsteuerung. Diese Referenz hat das Backend auch, sobald die Spielsteuerung die Methode *Anmelden* der Steuerungsanzeige aufruft: Beim Methodenaufruf übergibt die Spielsteuerung mit `this` eine Referenz auf sich selbst (siehe 15.3). Diese Referenz reicht die Steuerungsanzeige an das Backend weiter. Von da an hat das Backend eine Referenz auf die Spielsteuerung und kann Methoden der Spielsteuerung aufrufen, wenn ...
... sie weiß, wie diese Methoden heißen. Deshalb ist an dieser Stelle, d. h. in der Klasse SPIELSTEUERUNG, der Name der Methode nicht frei wählbar. Die Methode muss, wie im Backend vorgesehen, *AufTasteReagieren* heißen. Alternative Namen und Schreibweisen sind nicht möglich, sonst laufen die Aufrufe vom Backend ins Leere.

Damit die Methode *AufTasteReagieren* sinnvoll arbeiten kann, benötigt sie als Eingabewert die Information, welche Taste gedrückt wurde. Es gibt mehrere Möglichkeiten die „Tasteninformation“ zu kodieren[4]. So kann man die Tasten durchnummerieren oder jede Taste mit einem Text (String) beschreiben. Java bietet beides an.
In Java sind die Tastaturereignisse in der Java-Klasse KeyEvent beschrieben. Dort ist dem Drücken einer bestimmten Taste sowohl eine natürliche Zahl als auch eine Zeichenkette (String) zugeordnet. So wird beispielsweise das Ereignis des Drückens der Taste mit dem Buchstaben V als Zahl 86 kodiert.
Sinnvolle Tastaturereignisse für die Steuerung einer Spielfigur sind die Cursor-Steuerungs-Tasten links, oben, unten und rechts. Die folgende Tabelle zeigt, über welche Nummer bzw. welche Zeichenkette sie identifiziert werden:

Identifikations-nummer	Identifikationszeichenkette	Bedeutung	Tastenbeschriftung
37	VK_LEFT	links	←
38	VK_UP	oben	↑
39	VK_RIGHT	rechts	→
40	VK_DOWN	unten	↓

Eine ausführlichere Tabelle mit mehr Tasten kannst du im Anhang B finden.

Man muss sich nun entscheiden, ob man als Eingabewerte der Methode *AufTasteReagieren* Zahlen (d.h. die erste Spalte aus der Tabelle oben) oder Zeichenketten (d.h. die zweite Spalte der Tabelle) verwendet. Die Entscheidung hier fällt für Zahlen[5]. Damit ergibt sich folgender Methodenkopf für *AufTasteReagieren*:

```
void AufTasteReagieren(int taste)
```

Aufgabe 15.10
Überlege dir einen Methodenrumpf zur Methode *AufTasteReagieren, die die Bewegung von* Mampfi steuert.

4 „Ein Code ist eine eindeutige Abbildungsvorschrift, die jedem Zeichen oder Zeichenfolge aus einer Zeichenmenge **eindeutig** ein Zeichen oder eine Zeichenmenge aus einem möglicherweise anderen Zeichenvorrat zuordnet.“ Beispiele sind das Morsealphabet oder Handzeichen beim Tauchen.

5 Mit Zahlen, nicht aber mit Zeichenketten ist eine Mehrfachauswahl möglich (siehe die folgenden Seiten)

Eine richtige Vorgehensweise für die Tastatursteuerung von Mampfi sind mehrere bedingte Anweisungen im Rumpf der Methode *AufTasteReagieren*, wie es durch das folgende Struktogramm veranschaulicht ist:

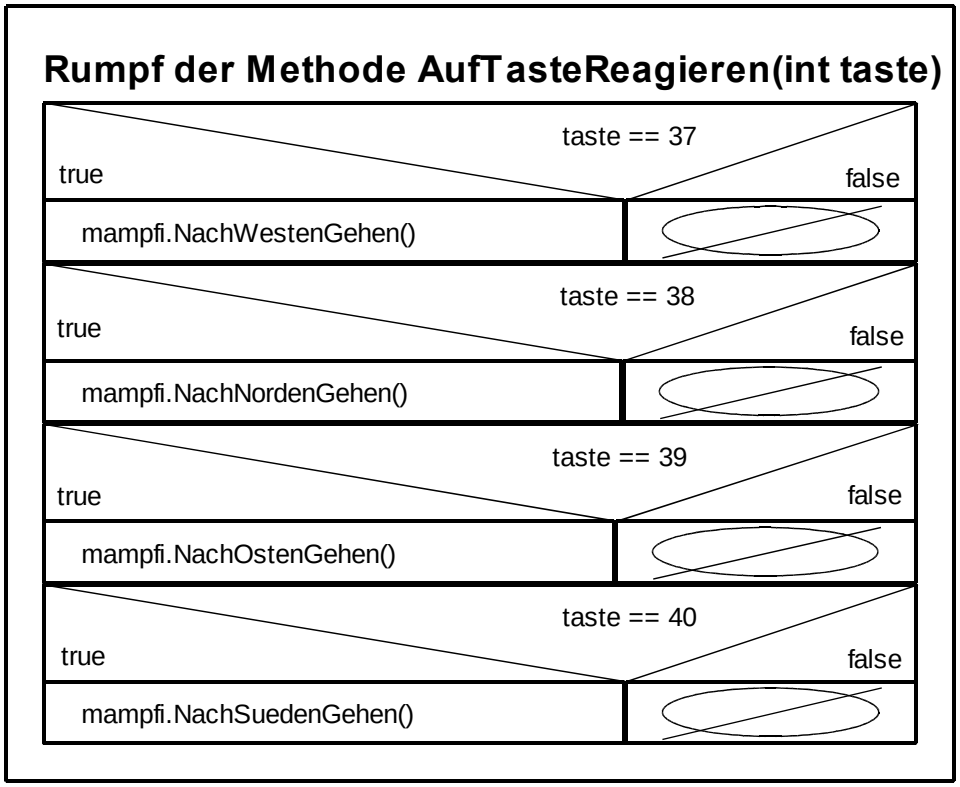


Abbildung 11: Rumpf der MethodeAufTasteReagieren — Bedingte Anweisungen sind nötig, denn abhängig vom Eingabewert soll eine andere Methode zur Bewegung ausgeführt werden.

In den Bedingungen aller bedingten Anweisungen wird die Übereinstimmung des Werts vom Eingangsparameter taste mit konkreten Zahlenwerten überprüft. Dieser Vorgang, der Vergleich

```
taste == ??
```

ist in allen vier bedingten Anweisungen immer der gleiche. Viele Programmiersprachen bieten für solche Fälle mit der **Mehrfachauswahl** eine kompakte und übersichtliche Kontrollstruktur an. In der Abbildung unten ist das zugehörige Struktogramm abgebildet. Hat der Eingangsparameter taste den Wert 37, so wird die Anweisung in der darunter liegenden Spalte ausgeführt. Hat taste den Wert 39, so ist es die Anweisung in der dritten Spalte.

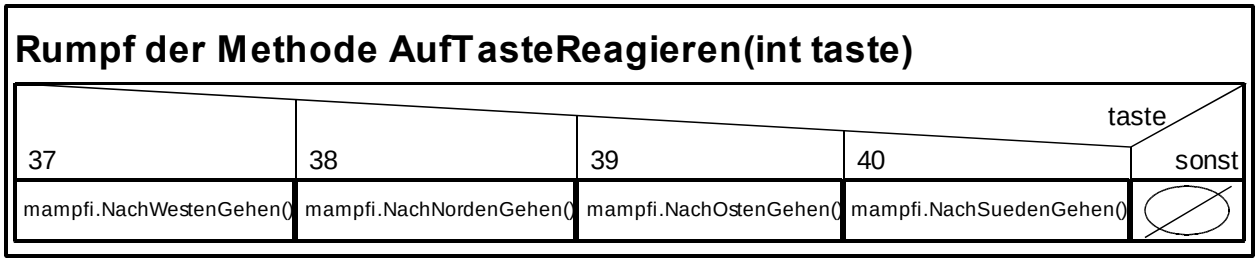


Abbildung 12: Alternative Formulierung des Rumpfs der MethodeAufTasteReagieren — Eine Mehrfachanweisung schafft Übersicht und reduziert den Quelltext.

Hinweise:
- Selbstverständlich kann in jedem Fall auch mehr als eine Anweisung ausgeführt werden.
- Wie in der bedingten Anweisung ist auch ein sonst-Fall möglich: Im Beispiel oben tritt dieser ein, wenn der Eingangsparameter **nicht** einen der Werte 37, 38, 39 oder 40 hat, dh. wenn eine andere Taste als eine der Cursor-Tasten gedrückt wurde. Da aber in dem Fall kein Handlungsbedarf besteht, bleibt die letzte Spalte leer, es wird keine Anweisung ausgeführt.

15.5 Tastatursteuerung endlich umsetzen – Mehrfachauswahl in Java

Die Umsetzung der Methode *AufTasteReagieren* mit Hilfe der Mehrfachauswahl sieht in Java wie folgt aus:

```
void AufTasteReagieren(int taste)
 {
      switch(taste)
     {
         case 37: // Taste "links"
             mampfi.NachWestenGehen();
             break;
         case 38: // Taste "oben"
             mampfi.NachNordenGehen();
             break;
         case 39: // Taste "rechts"
             mampfi.NachOstenGehen();
             break;

        // ...
     }
 }
```

Hinweis:
Das Schlüsselwort `break`[6] sorgt dafür, dass nur die Anweisungen innerhalb **eines** Falles (case) ausgeführt werden. Würde man im obigen Quelltext alle Schlüsselwörter break weglassen und hätte der Eingangsparameter den Wert 37, dann würde nicht nur der Methodenaufruf mampfi.NachWestenGehen() ausgeführt werden, sondern auch mampfi.NachNordenGehen(), mampfi.NachOstenGehen() usw.

Allgemein hat die Mehrfachauswahl in Java folgende Syntax:

```
switch (Auswahlkriterium)
{
     case wert1:
     //Anweisungen für Fall 1
     break;
     case wert2:
     //Anweisungen für Fall 2
     break;
     //...
     default:
     //Anweisungen für sonst
}
```

6 engl.: Abbruch; An dieser Stelle wird die case-Anweisung abgebrochen!

Hinweise:
- Die Anweisungen nach default werden nur ausgeführt, wenn es vorher keine Übereinstimmung gibt.
- Der Sonst-Zweig (default) kann bei der Mehrfachauswahl genauso weggelassen werden wie bei bei einer bedingten Anweisung (else-Zweig).
- Für das Auswahlkriterium akzeptiert Java von den bisher verwendeten Datentypen nur int und char, nicht aber String und float.

Aufgabe 15.11

a) Ergänze in der Klasse SPIELSTEUERUNG die Methode *AufTasteReagieren* wie oben besprochen. Verwende dabei die Mehrfachauswahl.

b) Teste die Methode ausführlich. Hast du ein Labyrinth mit mit Mauern kannst du nun sehr schnell testen, ob deine Bewegung vor der Mauer bzw. Spielfeldbegrenzung endet.

Aufgabe 15.12

Folgende Methode ist in der Klasse MAMPFI denkbar:

```
/**
    * Setzmethode des Attributs blickrichtung
    * @param blickrichtungNeu
    */
    public void BlickrichtungSetzen(char blickrichtungNeu)
    {
        if (blickrichtungNeu == 'S')
        {
            NachSuedenBlicken();
        }
        else
        {
            if(blickrichtungNeu == 'W')
            {
                NachWestenBlicken();
            }
            else
            {
                if(blickrichtungNeu == 'N')
                {
                    NachNordenBlicken();
                }
                else
                {
                    if(blickrichtungNeu == 'O')
                    {
                        NachOstenBlicken();
                    }
                    else
                    {
                          System.out.println("Falsche Eingabe in der Methode
                                              VerwundbarSetzen!");
                    }
                }
            }
        }
    }
```

Mit einer Mehrfachauswahl kann der Methodenrumpf deutlich kürzer und übersichtlicher formuliert werden. Schreibe die Methode BlickrichtungSetzen mit dem alternativen Methodenrumpf in dein Heft oder in dein BlueJProjekt.

15.6 Zusammenfassung

Aufgabe 15.13
In diesem Kapitel sind neu Zustandsdiagramme und die Mehrfachauswahl.
- Ergänze in deiner bisherigen Zusammenfassung die Bestandteile von Zustandsdiagrammen und füge ein selbst gewähltes Beispiel hinzu.
- Beschreibe weiterhin Struktur und Einsatzmöglichkeiten der Mehrfachauswahl. Ein Beispiel, wie diese Kontrollstruktur in Java umgesetzt wird. ist sicher auch hilfreich.

Aufgabe 15.14
Sicher ist es aus der Sicht eines neutralen Spielbetrachters naheliegend, dass die „oben“ Taste dafür sorgt, dass Mampfi einen Schritt nach Norden geht und die „rechts“ Taste, dass Mampfi einen Schritt nach Osten geht.
Versetzt man sich jedoch in Mampfi selbst, wäre es naheliegend, dass die „oben“-Taste dafür sorgt, dass Mampfi einen Schritt nach vorne geht (in die aktuelle Blickrichtung) und die „rechts“ Taste, dass Mampfi einen Schritt nach rechts geht. Die „unten“-Taste würde dann dafür sorgen, dass Mampfi einen Schritt zurück geht und sich dabei umdreht.
Speichere dein Projekt unter einem Namen ab und ändere dann die Methode *AufTasteReagieren* so um, dass die Steuerung der Bewegung von Mampfi wie gerade beschrieben aus der Sicht von Mampfi erfolgt.
Teste die Methode! Du wirst sehen, das ist etwas für anspruchsvolle Spieler!

Aufgabe 15.15 **Zweispielermodus**
Im Team spielen macht mehr Spaß! Erweitere die Klasse SPIELSTEUERUNG so, dass nicht nur ein, sondern zwei Mampfi erzeugt werden, die getrennt gesteuert werden können. In der Regel verwendet man für die Steuerung einer zweiten Spielfigur die Tasten „A“ (für links), „W“ (für oben), „D“ (für rechts) und „S“ (für unten),

Anhang A: Tipps zu den Aufgaben

Tipps zu Aufgabe 15.5:
Konstruktor der Klasse SPIELSTEUERUNG

```
public SPIELSTEUERUNG()
{
     // Attribute initialisieren
     punkteStand = 0;
     leben = 3;
     level = 1;
     // Labyrinth erzeugen und dem Referenzattribut zuweisen
     labyrinth = new LABYRINTH();
     // Mampfi erzeugen (mit dem eben erzeugten Labyrinth als
     // Eingabewert) und dem Referenzattribut zuweisen
     mampfi = new MAMPFI(labyrinth);
}
```

Anhang B Tastaturereignisse

Eine Auswahl der Tastaturereignisse in Java gibt die folgende Tabelle. Detailliertere Informationen sind in der Java-Dokumentation von Sun zu finden unter http://java.sun.com/j2se/1.5.0/docs/api/java/awt/event/KeyEvent.html

Identifikations-nummer	Identifikationszei-chenkette	Bedeutung	Identifika-tionsnummer	Identifikationszei-chenkette	Bedeutung
8	VK_BACK_SPACE	Backspace	67	VK_C	
10	VK_ENTER		68	VK_D	
19	VK_PAUSE		69	VK_E	
27	VK_ESCAPE		70	VK_F	
32	VK_SPACE	Leertaste	71	VK_G	
33	VK_PAGE_UP	Bild nach oben	72	VK_H	
34	VK_PAGE_DOWN	Bild nach unten	73	VK_I	
37	VK_LEFT	links	74	VK_J	
38	VK_UP	oben	75	VK_K	
39	VK_RIGHT	rechts	76	VK_L	
40	VK_DOWN	unten	77	VK_M	
44	VK_COMMA		78	VK_N	
45	VK_MINUS		79	VK_O	
48	VK_0		80	VK_P	
49	VK_1		81	VK_Q	
50	VK_2		82	VK_R	
51	VK_3		83	VK_S	
52	VK_4		84	VK_T	
53	VK_5		85	VK_U	
54	VK_6		86	VK_V	
55	VK_7		87	VK_W	
56	VK_8		88	VK_X	
57	VK_9		89	VK_Y	
65	VK_A		90	VK_Z	
66	VK_B				